Parallel unit propagation: Optimal speedup 3CNF Horn SAT

Uzi Vishkin

Abstract

A linear work parallel algorithm for 3CNF Horn SAT is presented, which is interesting since the problem is P-complete.

Introduction

The objective of this short paper is a linear work parallel algorithm for 3CNF Horn SAT. The algorithm achieves linear speedup over the best linear time serial algorithm up to some number of p processors on a PRAM. The result has special merit since the problem is P-complete, as explained in [1]. This makes it unlikely that there will be an NC (poly-log time using polynomial number of processors) algorithm for it and linear speedup not reaching poly-log time is that the best one can hope for. The exact result is O(N/p + h log N) for an input of N literals on a p-processor Arbitrary CRCW PRAM, where *h* is the length of the longest chain of dependencies in the input formula, as the main body of the loop will require *h* iterations (by analogy to BFS). This means that the best parallel time (known also as "depth") that the parallel algorithm can provide would be O(h log N), and it will need N/(h log N) processor to achieve that. Satisfiability problems have received extensive attention in the literature. Simon Kasif has published quite a few interesting papers on parallel complexity of various satisfiability instances, such as [3]. However, none is directly related to the current paper.

Basics

Input. 3CNF Horn SAT: in each clause: up to 3 literals and up to one of them is positive. Let N be the total number of literals.

Input form: Array of size K for variables P1,…,PK. Array of size m for clause C1,…,Cm. Each clause has a link to a subarray of size at most 3 comprising its literals. For each variable has a link to a subarray of its literals in the clauses. This is presented below as a bipartite graph with k+m nodes, one per variable, and one per clause. Each literal-clause pair provide an edge. We follow the standard input representation for parallel graph algorithms, such as in page 81 of [4]. See Figure 1.

Example: C1 = (x1 V ¬x2 V ¬x3), C2 = (x2 V ¬x3 V ¬x4), C3 = (x3 V ¬x1),

C4 = (x1)

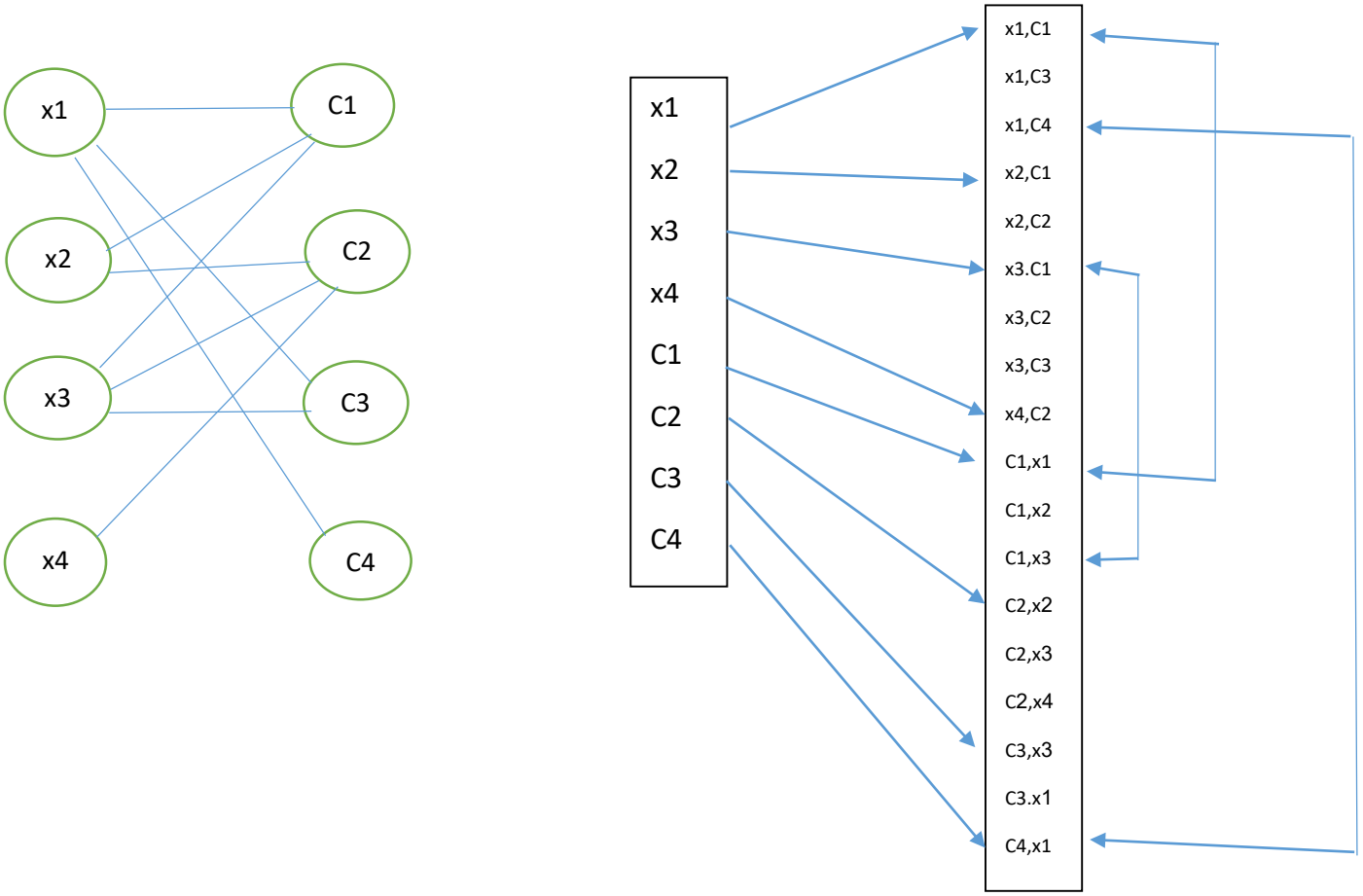

Figure 1. Left side: The bipartite graph for the 3CNF formula example. Middle: column: the vertex array: variable vertices followed by the clause vertices. Right column: The edge array. Each variable vertex has a pointer to the beginning of the subarray of its literals. Each clause vertex has a pointer to the beginning of its literals. Each edge in the bipartite graph appears twice: once for each of its vertices. Finally, each edge has a pointer to its other copy. Note: only an arbitrary subset of these pointers is shown.

The algorithm

Similar to [1] we assume that Horn formulae are in a "reduced" form, i.e., that there are no duplicate clauses and no duplicate literals within clauses.

The loop of the parallel algorithm (parallel unit propagation)

While there are clauses with singleton literals ("unit clauses") satisfy all of these literals, do {

In parallel, "Remove":

(i) all satisfied clauses, and

(ii) all implied FALSE literals in every clause in which they appear. For every affected clause, perform the following case analysis:

a. No literals remain. Conclude: formula is unsatisfiable.

b. Only one literal remained. Include this literal in the next iteration.

}

Final step. If the above did not conclude that the formula is unsatisfiable: assign FALSE to all remaining variable to derive a satisfying assignment.

Correctness. When there are no unit clauses upon reaching the final step, each remaining clause must contain at least one negative literal. Therefore, FALSE assignment to all their variables will satisfy all of these literals and their clauses.

Implementation comments

We assume the Arbitrary CRCW PRAM, in which p synchronous processors can read simultaneously any shared memory location, as well write simultaneously. In the case of several simultaneous writes into the same memory locations an arbitrary one succeeds.

The computation below applies balanced-binary-tree prefix-sum computations. For each such data structure we assume without loss of generality that the number of element (leaves in the tree) is always a power of 2, as it is always possible to round the number of leaves to the next power of 2 (achieving the same complexity).

In O(m) work and O(1) time find all clauses which are unit clauses. For each unit clause assign a satisfying (TRUE or FALSE truth value) to its variable. This may involve concurrent writes.

Apply prefix-sum over the K variables to assign serial numbers to all the just assigned variables.

Now, do a prefix sum computation over the degrees of these assigned variables to allocate a serial number to every one of their literals (within their respective clauses). This takes O(s) work and O(log s) time for s assigned variables. There are 4 cases once the literal of a clause gets its final truth value from its variable:

Case 1. The clause is satisfied. In this case, remove the clause.

When the literal is FALSE there are 3 cases:

Case 2. Two unassigned literals remain in the clause. Then do nothing.

Case 3. One unassigned literal remains. Namely, a new unit clause was found. Then the variable of that literal will be marked, possible involving concurrent writes. Using parallel prefix sum computation, the variable will be given a serial number and included in the next iteration.

Case 4. No unassigned literals remain. Then the whole formula is declared unsatisfiable.

To complete the loop, each occurrence of Case 3 above will induce a “thread” of activity. Inductively, the next iteration is now ready to begin.

In total, the algorithm requires O(N) work and O(h log N) time, where h is the length of the longest chain of dependencies in the input formula, as up to h iterations of the main loop of the algorithm above.

Acknowledgement

Moshe Vardi suggested this problem after seeing [2].

References

1. Dowling, William F. and Gallier, Jean H. (1984), "Linear-time algorithms for testing the satisfiability of propositional Horn formulae", Journal of Logic Programming, **1** (3): 267–284, doi:10.1016/0743-1066(84)90014-1, MR 0770156
2. Edwards J. and Vishkin U. (2021). “Study of Fine-Grained Nested Parallelism in CDCL SAT Solvers”. ACM Transactions on Parallel Computing (TOPC), to appear.
3. Kasif, S. On the parallel complexity of discrete relaxation in constraint satisfaction networks. Artificial Intelligence 45 (3), 275-286, 1990
4. Vishkin, U (2010). Thinking in Parallel: Some Basic Data-Parallel Algorithms and Techniques, class notes, 2010, 104 pages. http://users.umiacs.umd.edu/~vishkin/PUBLICATIONS/classnotes.pdf